ECSE426 - Microprocessor Systems
Group 20 - Lab 3 & 4

Fabrice Normandin 260636800
Marcel Morin 260605670

March 20, 2018

Abstract

This report outlines the engineering process we used while completing Labs 3 & 4. These most recent labs built upon the ADC sampling, filtering and 7-segment display work done previously as part of Lab 2. The main objective of this lab was to create a system which accepts user input through a keypad, and then attempts to match the given value (a RMS voltage) by using a PWM signal. This system also had to successfully manage the transitions between the different states of operation (sleep, keypad input, voltage matching), all-the-while conserving CPU resources as much as possible. The system was first created using an interrupt-driven approach (Lab3), and later using the multithreaded paradigms of the CMSIS_RTOS framework (Lab4). Despite some portions of this lab presenting a significant technical challenge, it can be considered a resounding success, as all of its objectives were successfully completed. While our current solution makes a great effort towards CPU efficiency, additional measures could be taken in order to reduce the power consumption even further.

Todo list

Contents

1 Problem Statement

A comprehensive list of all system requirements can be found within the Lab 3 & Lab 4 handouts. Among these requirements, one of the most challenging to meet was one related to the keypad, as it showed that the system had to respond differently to buttons depending on their press duration, and generate a transition between different modes of operation (a long “” press would put the system into sleep mode, for example). Consequently, an additional requirement was created, by which the system code had to be structured as a finite-state machine (FSM), and successfully manage state transitions in a clear, well-structured way. State diagrams were also deemed essential, and we considered it a system requirement that each sub-system have its own related state diagram, in order to clearly represent the functioning of our system.

2 Theory and Hypothesis

The main component of this experiment is the Pulse Wave Modulation (PWM) which would apply a voltage across a load and a capacitor. The PWM has a set cycle time that we picked to be faster than the time constant of our circuit. The equation bellow is used to calculate the time constant of an RC circuit.

$$\tau = RC$$

During this cycle we can choose the duty cycle, or the time the voltage is turned on during the total cycle time of our PWM unit. With our capacitor and resistive load in place the capacitor would collect charge during the active duty cycle of our PWM pulse, and then dissipate its charge during the rest of the PWM cycle. Would the duty cycle of the PWM start at 0, there would be no active high voltage outputted by the timer during the cycle. This would give a calculated RMS value of 0 volts. By increasing the duty cycle of the PWM pulse we can start to introduce a voltage for a portion of the PWM cycle. This would charge the capacitor and have it then discharge on the off duty cycle time, leading to a larger RMS value. Figure ? bellow shows the charging of the capacitor during the active duty cycle of our rectifier circuit.

add picture here

With our PWM timer created, we need a controller to compare our measured RMS with the target RMS value given by the user via keypad. This controller would then see if the duty cycle needed to be increased or decreased based on if the measured RMS value was above or bellow our target. The implementation of this controller would be quite simple as it just needs to compare two values in order to determine which action to perform.

```
if(calculated_RMS < target_RMS){
        HAL_TIM_PWM_ConfigChannel(++); //increase Duty Cycle
} else{
        HAL_TIM_PWM_ConfigChannel(--); //decrease Duty Cycle
}
```

Due to the frequency we picked for our PWM pulse, we needed to pick an appropriate ADC value to ensure that we were picking up the changes applied by our controller. Our ADC frequency needs to be lower than that of our PWM timer so that the ADC samples seemingly random voltages in Figure ? above. With multiple random samples taken across multiple PWM cycles our converted ADC values will give an accurate combined RMS calculation of our collected data points. With our resistor and capacitor having values $4.7k\Omega$ and $0.1\mu F$ the time constant is calculated to be $\tau = (4.7k\Omega)(0.1\mu F) = 0.470ms$. Therefore picking a PWM timer frequency of $10kHz$ which gives us one clock cycle every $0.1ms$ which is faster than the time constant needed for our controller to receive an accurate RMS value from our ADC. To pick an ADC value we just need to follow our constraint above that the ADC have a lower frequency than the PWM timer.

$$Timer freq. = \frac{Clock freq.}{(prescaler + 1) * period} \tag{1}$$

With a prescaler of 83 and period of 1000 the Timer frequency of the ADC comes out to be $1kHz$ which falls within our system constraints.

The second aspect of this lab was to implement threads into our system. We did this using the FreeRTOS (Real Time Operating System) kernel. This would make available a middle ware that would be able to schedule and handle the creations of threads and free up our CPU. The advantages of running multiple threads concurrently means that our display thread (which takes care of refreshing our display with the appropriate user input value or measured RMS value of our circuit) and our keypad thread can coexists. Another advantage of running threads in our system is that we are no longer dependent off of the internal Systick clock. The ADC thread is also something that we considered implementing with raising a flag once our DMA buffer was filled that would wake up a thread to compute the RMS value. Whether to implement this in a thread or ISR is equivalent and will depend on the implementation and testing of our system.

3 Implementation

This section will describe the design and engineering processes that were used while implementing our system, based on the requirements described above. In order to simplify and structure its description, the system will first be broken down into its different logical components.

3.1 High-Level System Architecture

The various requirements and corresponding program functionality of our system can be broken down into four main areas: the 7-segment display, the keypad, the ADC and the PWM controller. The keypad and display components were implemented as Threads, each behaving as a simple finite-state-machine. The ADC and PWM-related logic components were implemented as a part of an Interrupt-Service-Routine (ISR).

Despite the overall system having significant complexity, its high-level behaviour can be broken down into a very limited number of states. The transitions between each state were also well defined (as part of the Section 1). By taking advantage of this fact, a very simple state machine was created, where each state manages to turn each sub-component "on" or "off". A state diagram showing the three system states, along with the name of their transitions and their outputs can be seen in Figure 1.

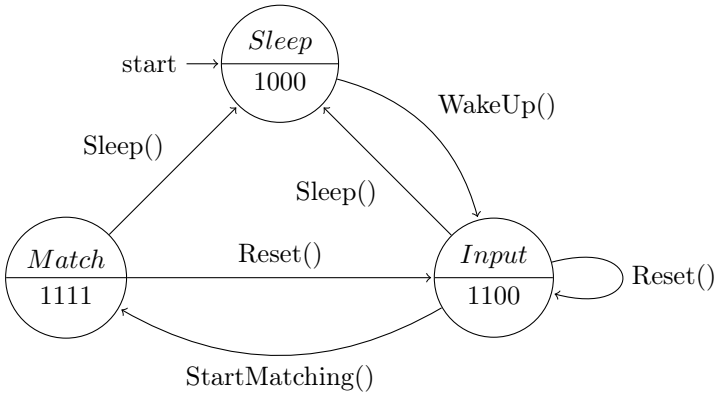

Figure 1: System high-level state diagram

Sleep	Input	Match
The sleep state is the initial state of the system. In this state, the only active component is the keypad thread. All other components (display, ADC, PWM timer) are turned off, in order to conserve energy. Whenever a keypress is detected, the system transitions into the 'input' state.	During the "input" state, the Keypad thread detects button presses and updates the target value accordingly, while the Display thread shows the current target value. The Keypad and Display threads are active, while the ADC and PWM timer are turned off. Whenever a valid voltage value is entered, the system transitions into the "match" state.	When the system enters the "match" state, the PWM timer and ADC are both started. After each `HAL_ADC_ConvCpltCallback()` interrupt service routine, the ADC samples are filtered and their RMS is extracted. This value is fed to the PWM controller, which modifies the period of the PWM timer in order to attempt to reach the target voltage.

The associated code for this finite state machine can be found within the fsm.c file. Every major component of the system will now be examined individually.

3.2 Display Thread

The logic used in the Display thread was almost all created as part of Lab 2. The main logic can be found within the `refresh_display()` function, within the display_thread.c file.

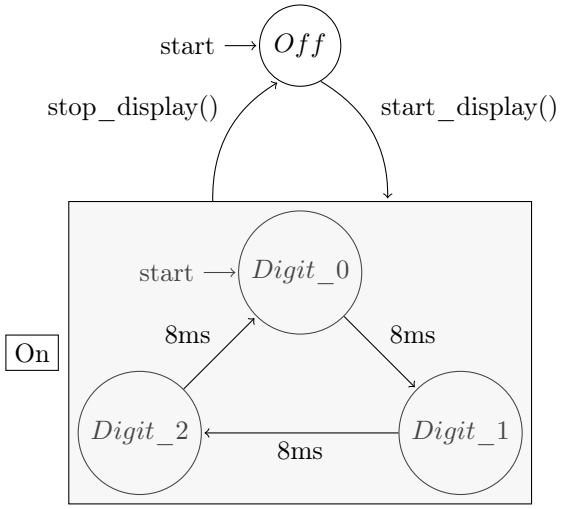

Figure 2: Display thread state diagram

3.3 Keypad Thread

3.4 ADC Logic

3.5 PWM Timer

4 Testing and Observations

5 conclusion